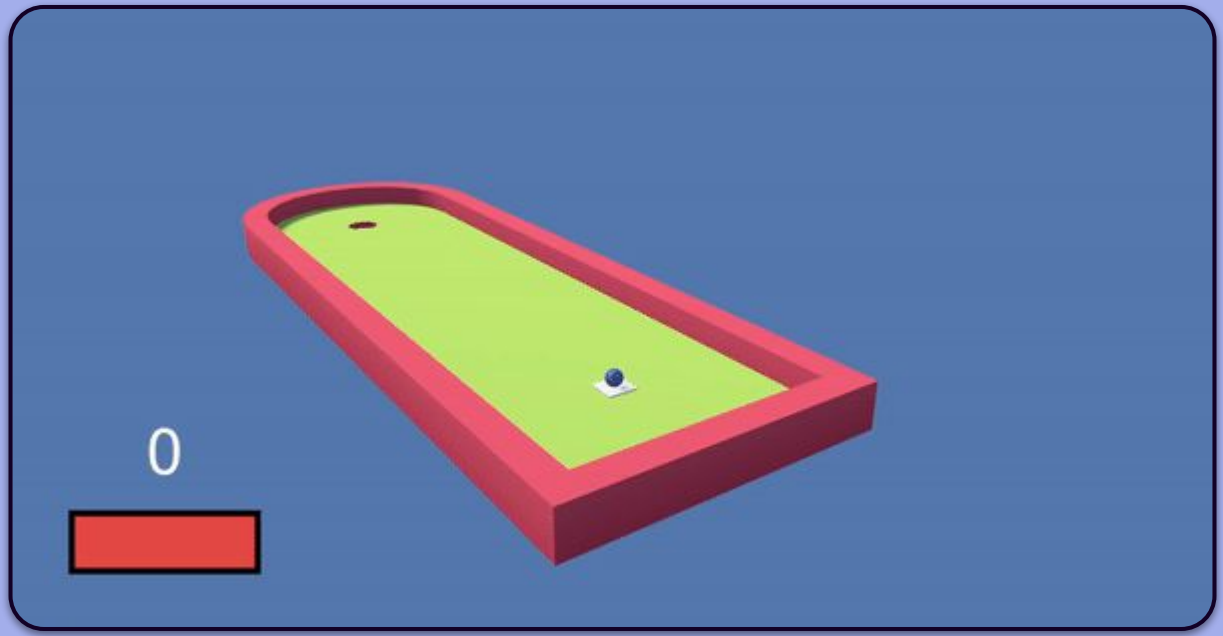
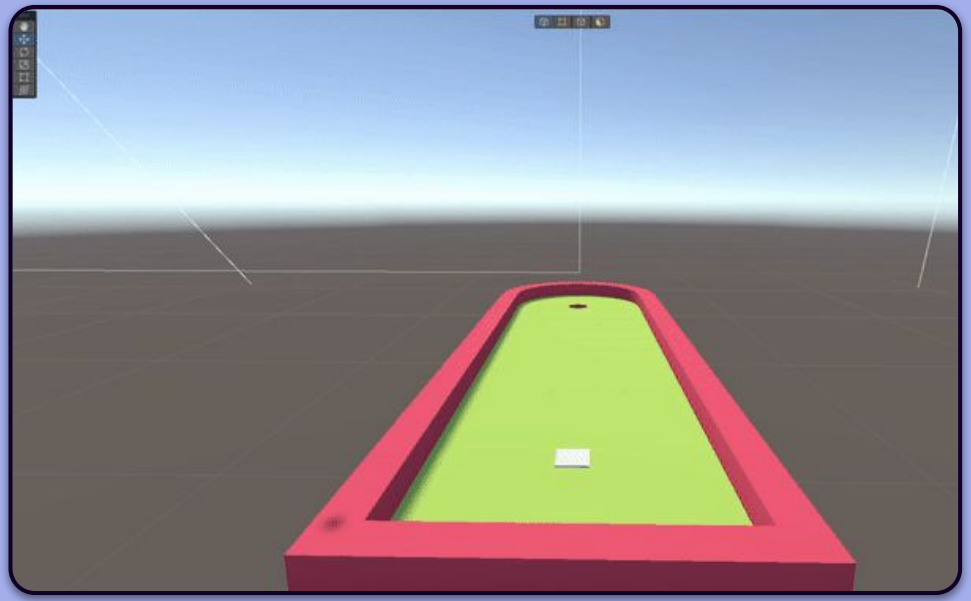
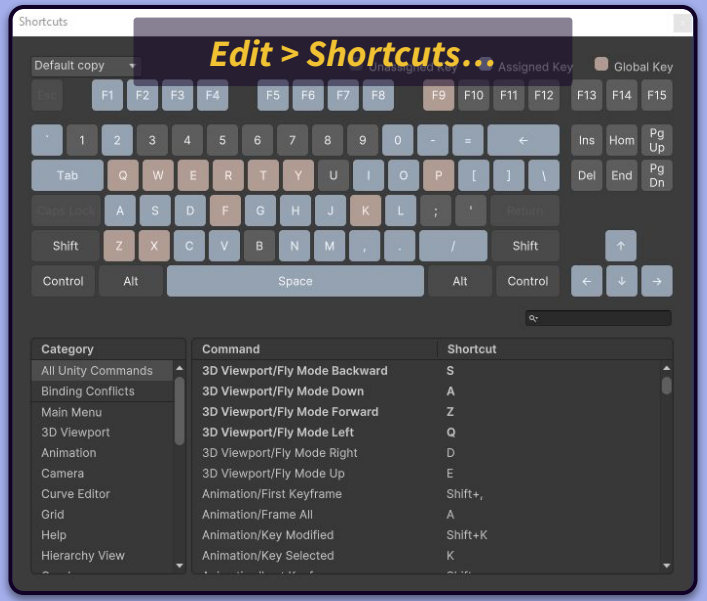
Créer sa première application 3D sur Unity



Les rudiments d'un environnement 3D

en débutant par le déplacement de la caméra et les raccourcis indispensables





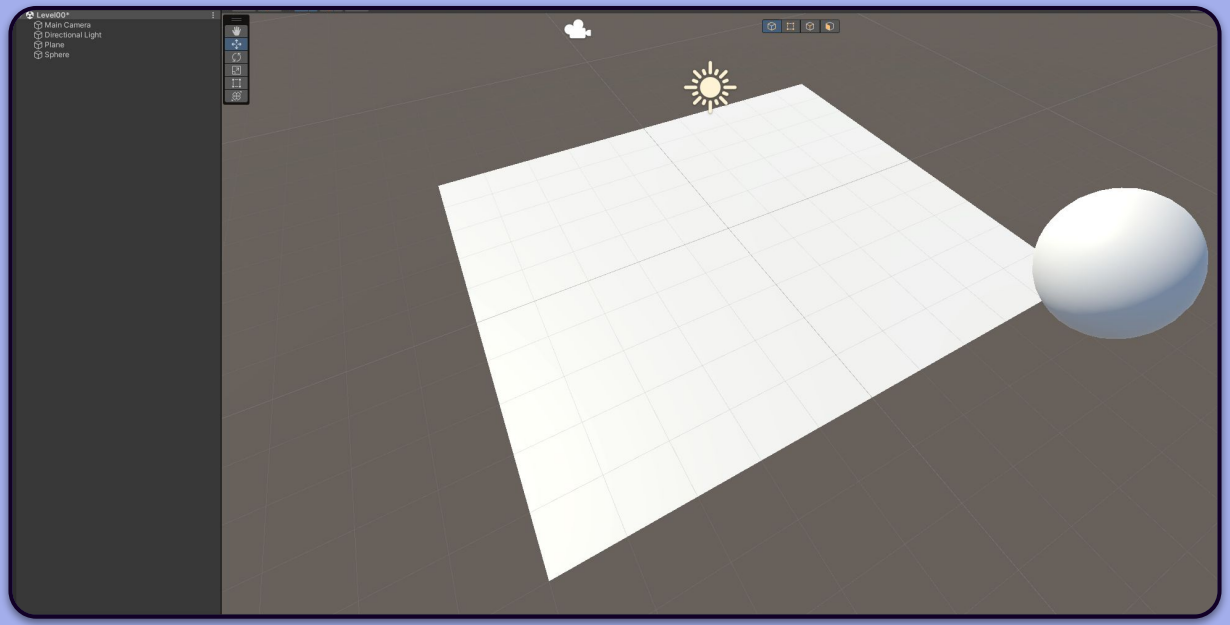
Clic droit pour réaliser une rotation de la caméra

Maintenir le *clic droit* et naviguer dans la scène avec les touches Z, Q, S, D

Maintenir le *clic droit* puis monter ou descendre la caméra avec les touches A et E

Créer des objets 3D

clic droit dans la hiérarchie > *3D Object*

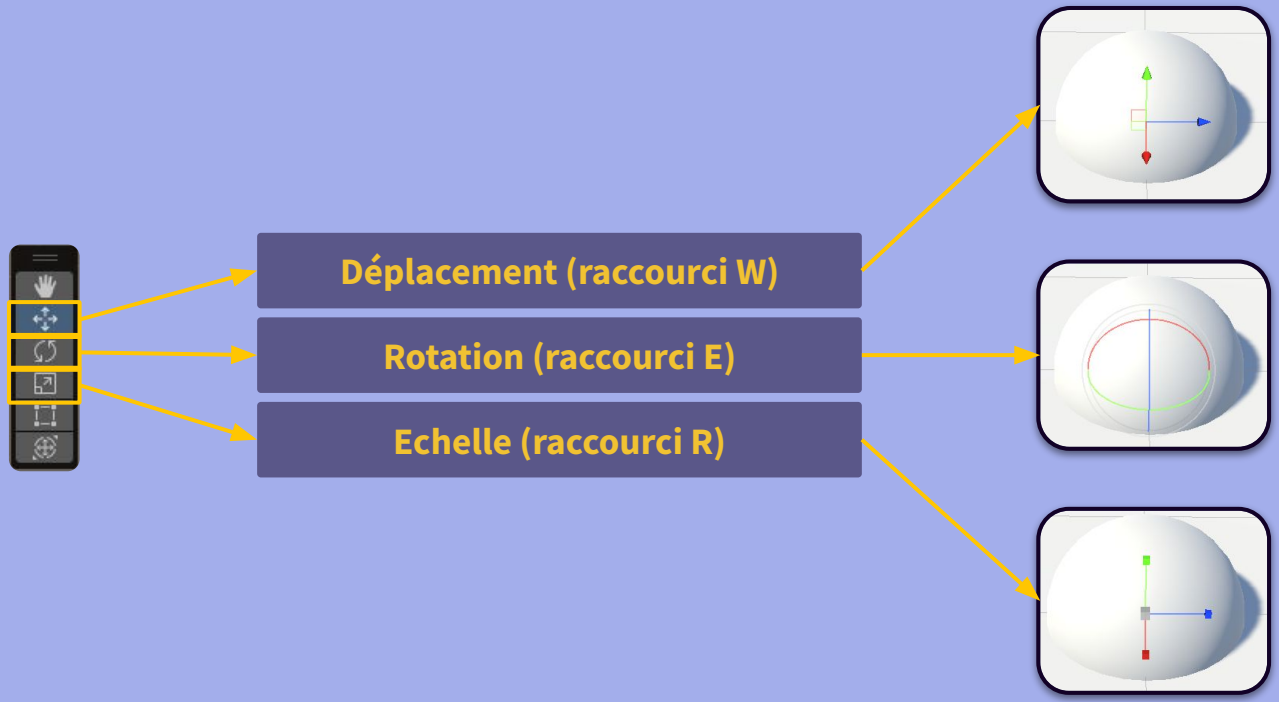


Créer une *Plane* et une *Sphere*

Réaliser un *Reset* de leurs coordonnées (*clic droit* sur leur *Transform* > *Reset*)

Modification des objets 3D dans la scène

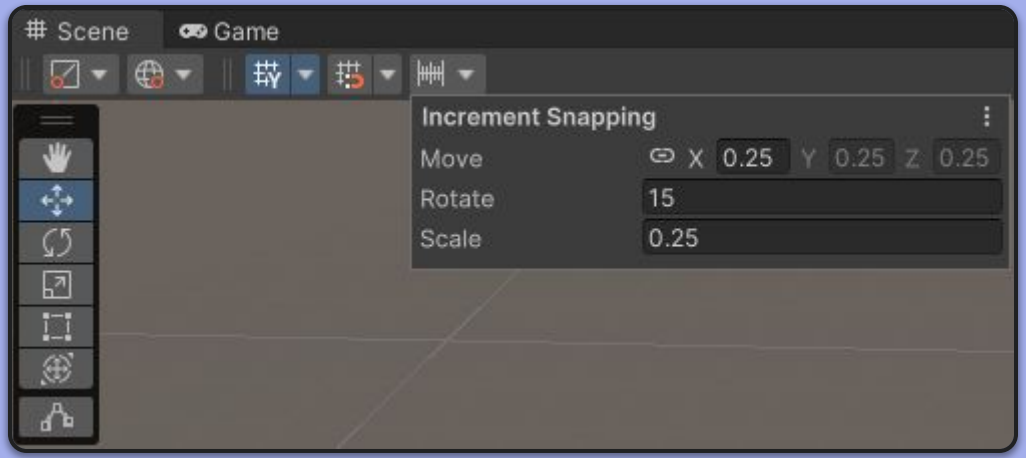
déplacement, rotation et échelle de l'objet



Vous pouvez modifier ces propriétés propriétés sur les axes x, y et z directement dans la scène (en manipulant les objets) ou alors en modifiant leur composant *Transform*

Modification des objets 3D dans la scène

déplacement, rotation et échelle de l'objet



A noter que si vous appuyez sur la touche *CTRL* tout en modifiant une des précédentes propriétés (position, rotation ou échelle) alors vous pouvez spécifier une valeur d'écart précise

Paramétrer la *Sphere*

ajouter un *Rigidbody* et un nouveau script pour la contrôler (*BallController*)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BallController : MonoBehaviour
{
    public float baseSpeed = 10f;
    public float maxTimeSpaceKeyDown = 3f;

    private Rigidbody _rb;
    private Vector3 _applyForce;
    private bool _spaceDown = false;
    private bool _spaceUp = false;
    private float _spaceKeyDownDuration = 0f;

    // Start is called before the first frame update
    void Start()
    {
        _rb = GetComponent<Rigidbody>();
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            _spaceDown = true;
        }

        if (_spaceDown)
        {
            _spaceKeyDownDuration += Time.deltaTime;
        }

        if (Input.GetKeyUp(KeyCode.Space) || _spaceKeyDownDuration > maxTimeSpaceKeyDown)
        {
            _spaceUp = true;
            _spaceDown = false;

            _applyForce = Vector3.forward * baseSpeed * _spaceKeyDownDuration;
        }
    }

    void FixedUpdate()
    {
        if (_spaceUp)
        {
            _rb.AddForce(_applyForce, ForceMode.Impulse);
            _spaceUp = false;
            _spaceKeyDownDuration = 0f;
        }
    }
}
```

Le déplacement de la *Sphere* (notre balle de golf) est basé sur un appui prolongé de la barre espace, plus j'appuie longtemps et plus la force transmise à la balle sera grande

Force de base qui sera appliquée à la balle et durée maximum pour appuyer sur la barre espace

Capturer la durée de l'appui

Calculer le *Vector3 Force* appliqué à la balle
direction (0, 0, 1) * force de base * durée de l'appui

Appliquer un vecteur *Force* sur le *Rigidbody* de notre balle

Appliquer une force à un *Rigidbody*

au sein de la fonction *FixedUpdate* (indépendant des *FPS* de l'utilisateur)

```
_rb.AddForce(_applyForce, ForceMode.Impulse)
```

La force appliquée sur le *Rigidbody* de notre balle modifie sa vitesse (*velocity*) de façon différente en fonction du mode utilisé :

ForceMode.Force interprète le vecteur d'entrée comme une force mesurée en Newton (N). La vitesse est modifiée de la façon suivante :

force * DT / masse

avec *DT* la valeur de *Time.fixedDeltaTime* (20ms soit 50 appels par seconde par défaut)

ForceMode.Acceleration interprète le vecteur d'entrée comme une accélération mesurée en mètre par seconde carré ($m \cdot s^{-2}$). La vitesse est modifiée de la façon suivante :

force * DT

ForceMode.Impulse interprète le vecteur d'entrée comme une impulsion mesurée en Newton par seconde. La vitesse est modifiée de la façon suivante :

force / masse

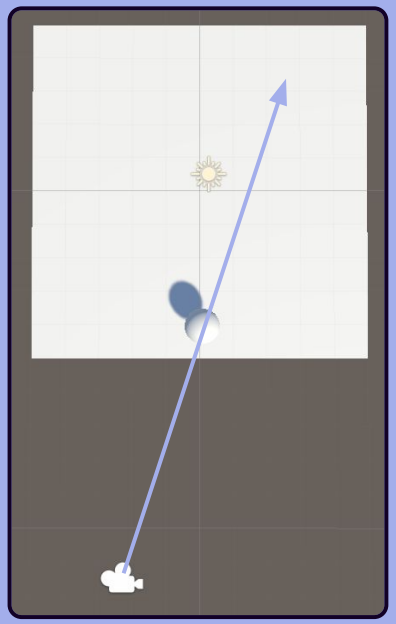
ForceMode.VelocityChange interprète le vecteur d'entrée comme un changement direct de vitesse (*velocity*) mesuré en mètre par seconde. La vitesse est modifiée par la valeur de force.

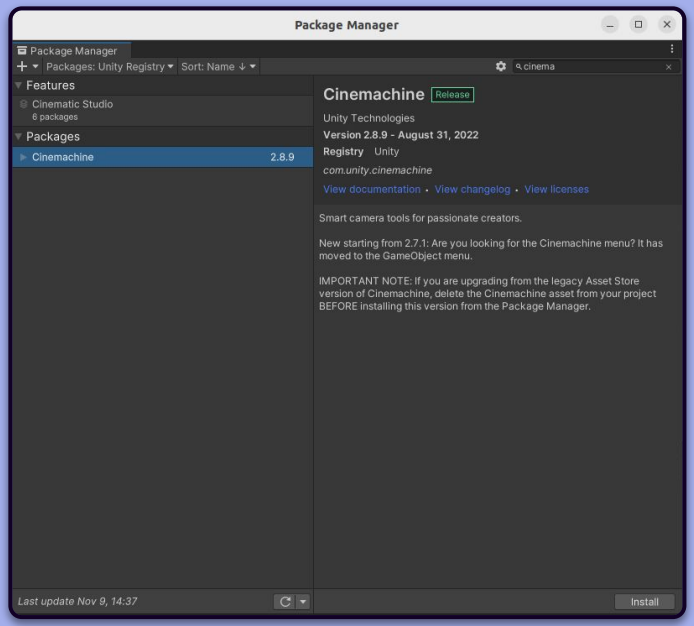
https://docs.unity3d.com/ScriptReference/Rigidbody.AddForce.html

Déplacer la caméra autour de la balle de golf

en utilisant le *package Cinemachine ; Windows > Package Manager*

L'objectif est d'être capable d'indiquer une direction précise lorsque l'on envoie notre balle, cette direction va correspondre au prolongement de l'axe entre notre caméra et notre balle



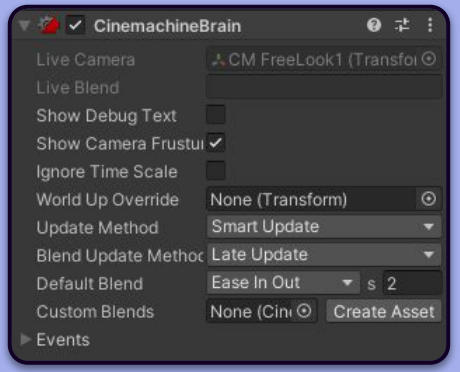


Ajouter une nouvelle *Cinemachine*

clic droit dans la hiérarchie > *Cinemachine* > *FreeLook Camera*

Le *package Cinemachine* est une suite d'outils conçue pour utiliser des caméras dynamiques et intelligentes

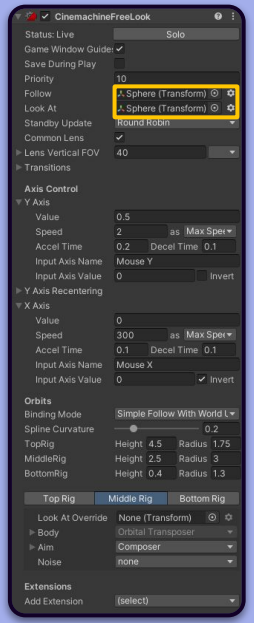
FreeLook Camera ajoute un nouveau *GameObject* avec un composant du même nom et ajoute un composant supplémentaire à notre caméra principale

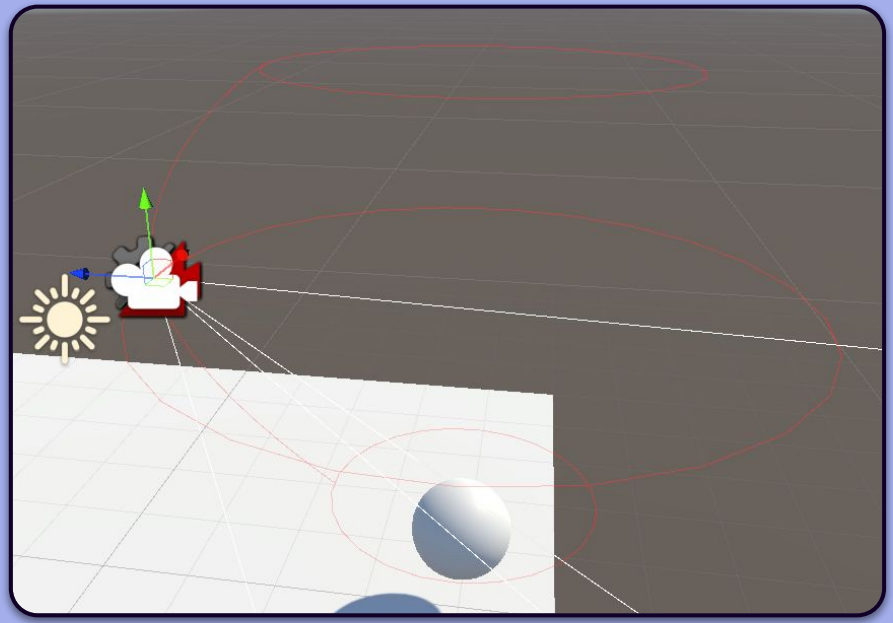


Les mouvements de notre caméra principale vont être dictés par la *FreeLook Camera*, nous épargnant ainsi de devoir coder la rotation de la caméra autour de notre balle

Paramétrer notre *FreeLook Camera*

glisser-déposer notre *Sphere* au sein des propriétés *Follow* et *Loot At* de notre *Cinemachine*





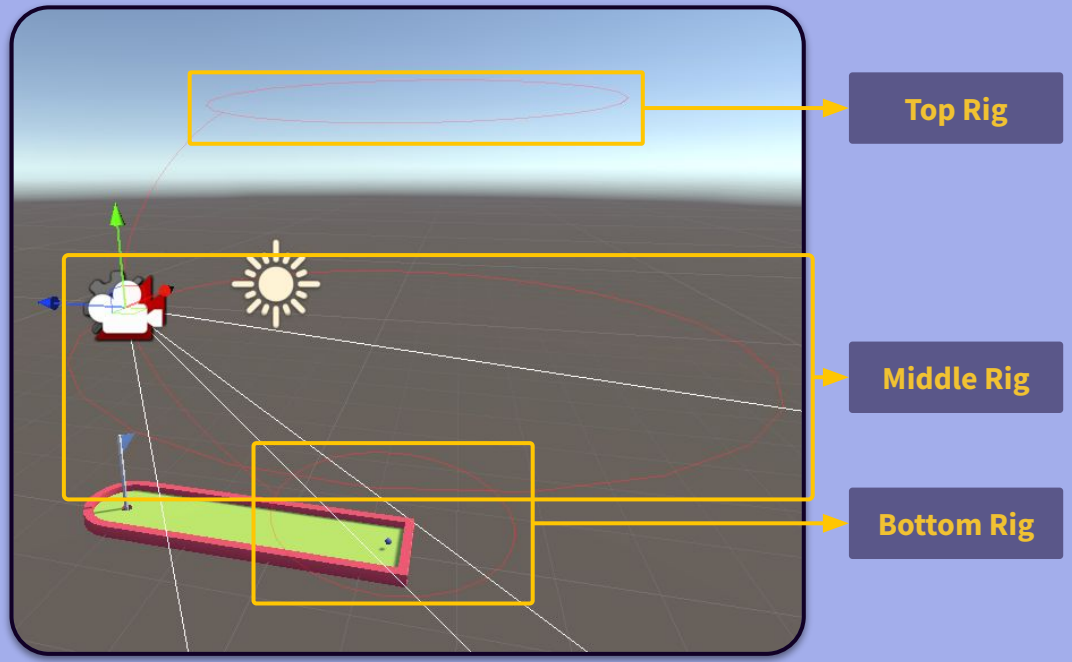
FreeLook Camera permet de gérer facilement des caméras en orbite autour d'un *GameObject*

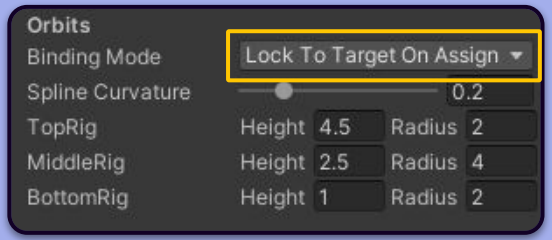
Les anneaux rouges symbolisent les limites de déplacement de la caméra

Paramétrer notre *FreeLook Camera*

en modifiant la hauteur et le rayon des anneaux

Le paramétrage de ces anneaux va permettre de plus ou moins restreindre les mouvements possibles de notre caméra face aux mouvements de notre souris






```
1  private void Awake()
2  {
3      Cursor.lockState = CursorLockMode.Locked;
4  }
```

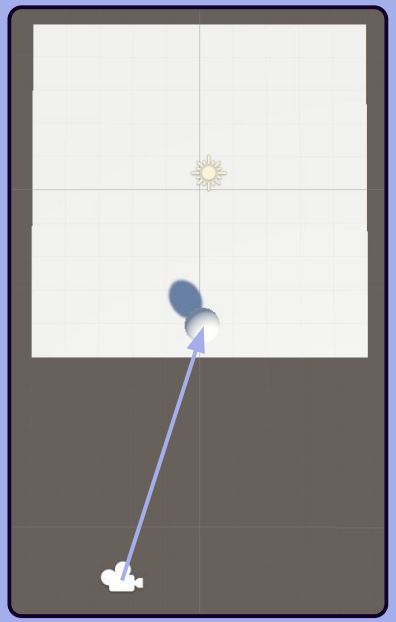
Ajouter cette fonction au script *BallController* pour masquer la souris lors des tests (*NB*: touche echap pour la récupérer)

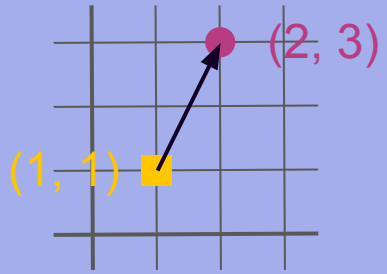
https://docs.unity3d.com/Packages/com.unity.cinemachine@2.8/manual/CinemachineBindingModes.html

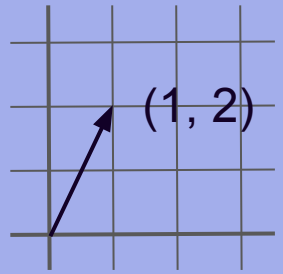
Modifier notre script *BallController*

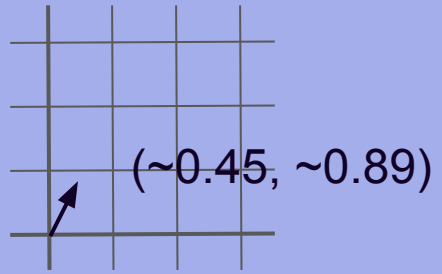
afin de tenir compte de la position de notre caméra par rapport à notre *Sphere*

Récupération du vecteur direction correspondant à la normalisation du vecteur entre notre caméra et notre balle









(2, 3) - (1, 1) = (1, 2)

Normalisation du vecteur (magnitude = 1)

$v / |v| = (1, 2) / \text{sqrt}(1^2 + 2^2) = (1 / \text{sqrt}(5), 2 / \text{sqrt}(5))$

Modifier notre script *BallController*

afin de tenir compte de la position de notre caméra par rapport à notre *Sphere*

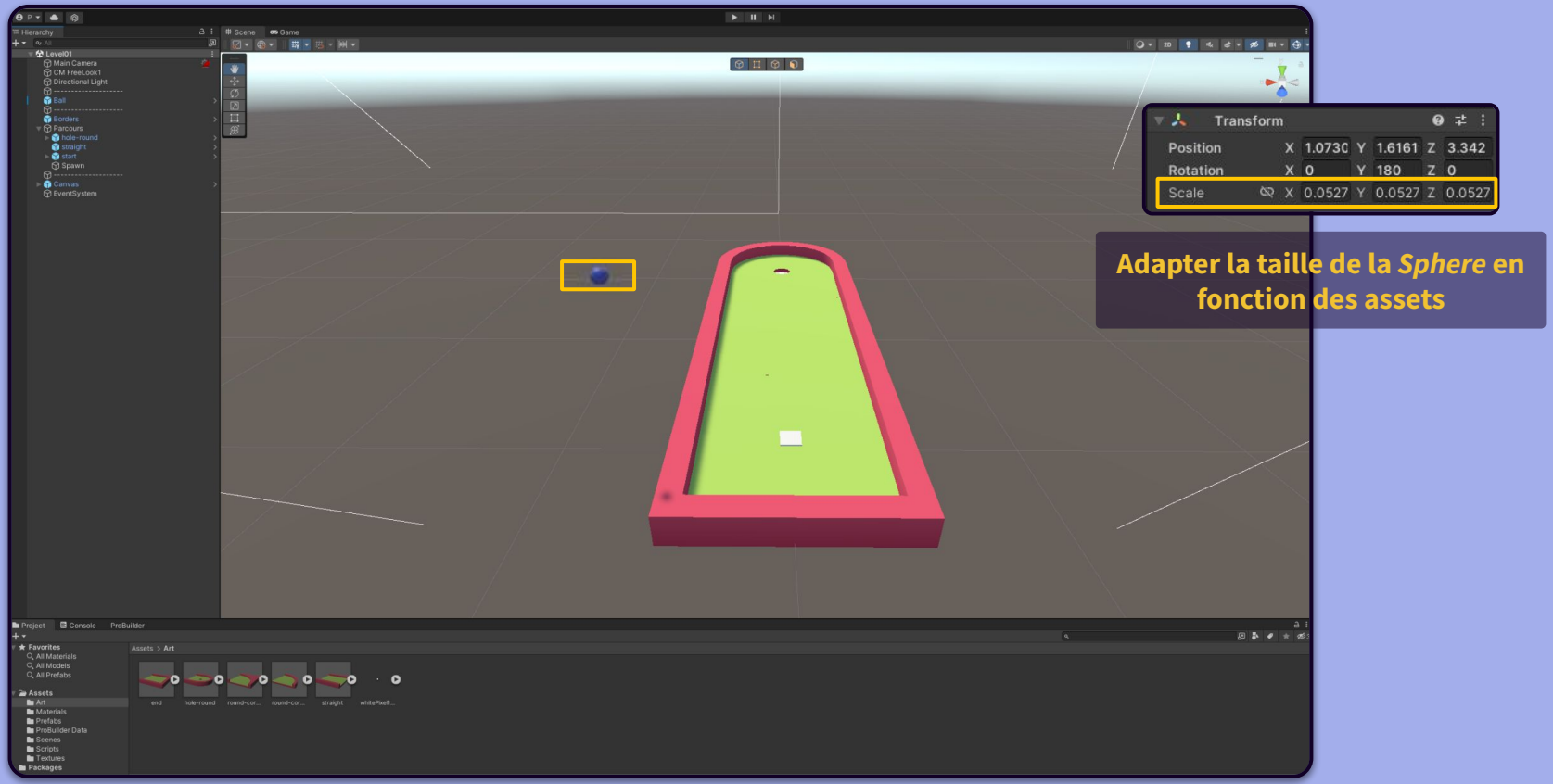
Modification de la fonction *Update* de notre script *BallController.cs*

```
1  if (Input.GetKeyUp(KeyCode.Space) || _spaceKeyDownDuration > maxTimeSpaceKeyDown)
2  {
3      _spaceUp = true;
4      _spaceDown = false;
5      Vector3 dir = (this.transform.position - Camera.main.transform.position).normalized;
6      _applyForce = dir * baseSpeed * _spaceKeyDownDuration;
7      _applyForce.y = 0;
8  }
```

Référence au tag *MainCamera*

Construction de l'environnement

importer les *assets* nécessaires (sans oublier la texture) et reproduire la scène suivante



Création des *Prefabs* de nos *assets*

afin de leur associer un *Mesh Collider*

Ajouter un composant *Mesh Collider* à vos *assets* puis les glisser-déposer dans un dossier *Prefabs* (lors de l'enregistrement de la *Prefab*, cliquer sur "*Original Prefabs*")

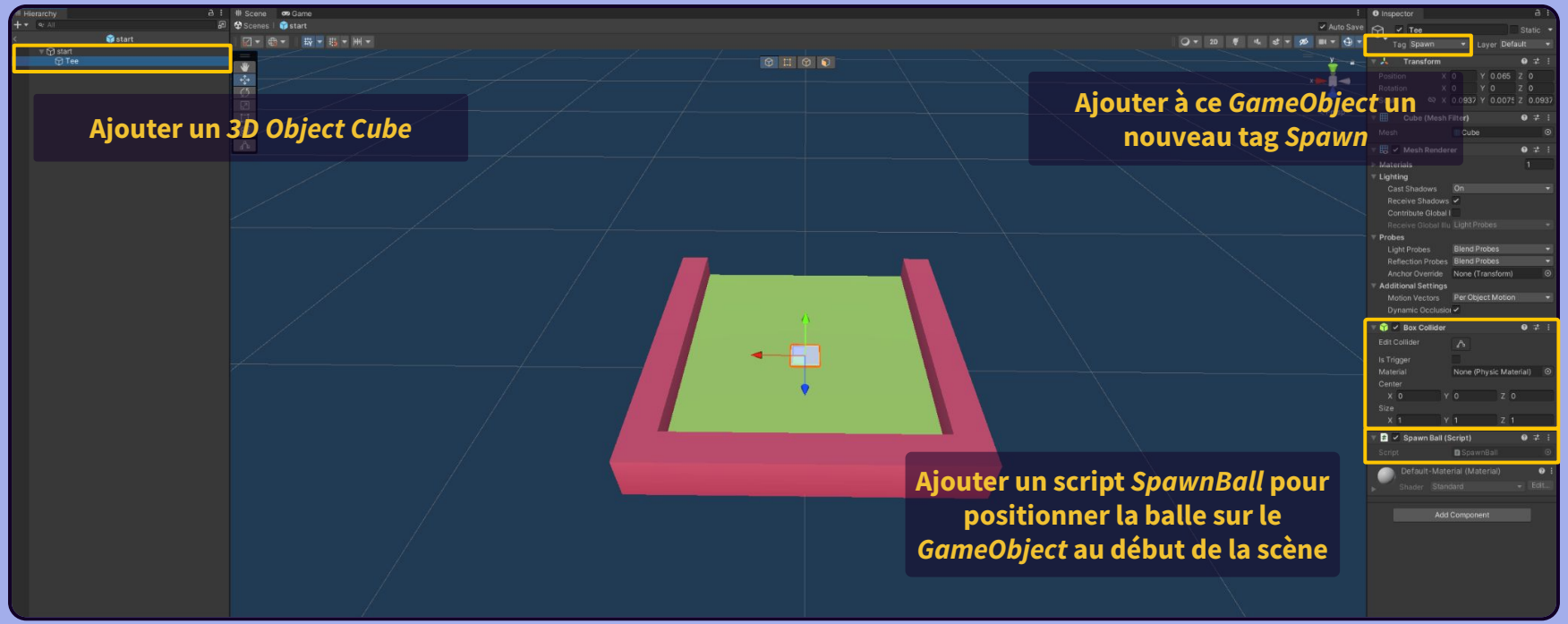
Modification de la *Prefab “hole-round”*

afin de leur associer d’autres *GameObjects* ou composants

Ce *GameObject Hole* va nous permettre de détecter lorsque la balle arrive dans le trou

Modification de la *Prefab “end”*

afin de leur associer d’autres *GameObjects* ou composants



Ce *GameObject Tee* va nous permettre de représenter le début du parcours

Script *SpawnBall*

pour positionner la balle au début de la scène

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class SpawnBall : MonoBehaviour
6  {
7      private GameObject _player;
8
9      // Start is called before the first frame update
10     void Start()
11     {
12         _player = GameObject.FindGameObjectWithTag("Player");
13         _player.transform.position = new Vector3(transform.position.x, transform.position.y + 0.25f, transform.position.z);
14     }
15 }
```

Associer le tag “Player” à la *Sphere*

Événement de collision et gestion des scènes

via la fonction *OnTriggerEnter* et le *package UnityEngine.SceneManagement*

```
1  using UnityEngine.SceneManagement;
```

Le *package UnityEngine.SceneManagement* permet de manipuler les scènes (chargement, index)

```
1  void OnTriggerEnter(Collider col)
2  {
3      if (col.CompareTag("Hole"))
4      {
5          int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
6          if (currentSceneIndex + 1 < SceneManager.sceneCountInBuildSettings)
7          {
8              SceneManager.LoadScene(currentSceneIndex + 1);
9          }
10     }
11 }
```

Le paramètre *col* de la fonction récupère l'élément avec lequel notre balle est rentrée une collision

Modification de notre script *BallController.cs*

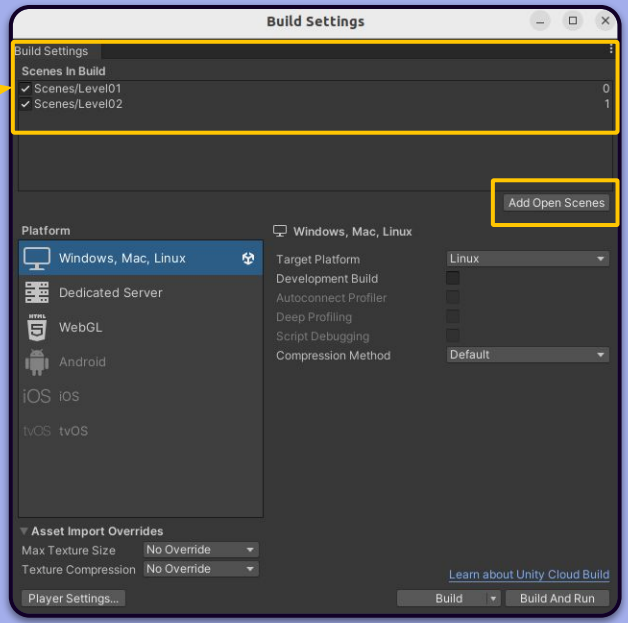
Événement de collision et gestion des scènes

via la fonction *OnTriggerEnter* et le *package UnityEngine.SceneManagement*

```
1  void OnTriggerEnter(Collider col)
2  {
3      if (col.CompareTag("Hole"))
4      {
5          int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
6          if (currentSceneIndex + 1 < SceneManager.sceneCountInBuildSettings)
7          {
8              SceneManager.LoadScene(currentSceneIndex + 1);
9          }
10     }
11 }
```

Récupération de l'index actuel de notre scène

Nous vérifions s'il existe une prochaine scène à charger



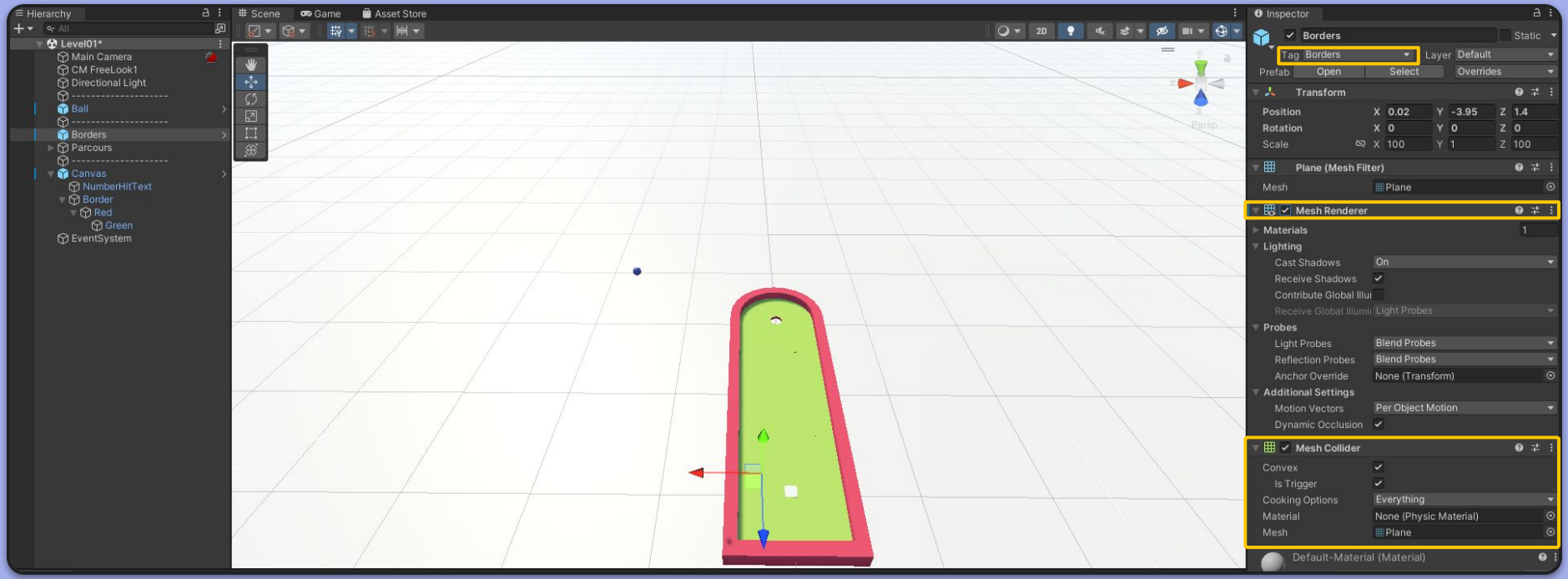
Gérer le cas dans lequel la balle quitte le terrain

en repositionnant la balle sur le *GameObject Tee*

En vous basant sur le code précédent et le tag *Spawn*, trouver une solution pour repositionner la balle sur le *GameObject Tee* lorsqu'elle quitte le terrain

Gérer le cas dans lequel la balle quitte le terrain

en repositionnant la balle sur le *GameObject Tee*



Créer un cube avec un *Mesh Collider* et le tag *Borders* (vous pouvez désactiver le composant *Mesh Renderer*

Gérer le cas dans lequel la balle quitte le terrain

en repositionnant la balle sur le *GameObject Tee*

```
1  private GameObject _spawnPoint;
2
3  void Start()
4  {
5      _spawnPoint = GameObject.FindGameObjectWithTag("Spawn");
6  }
```

Référence à notre *GameObject Tee*

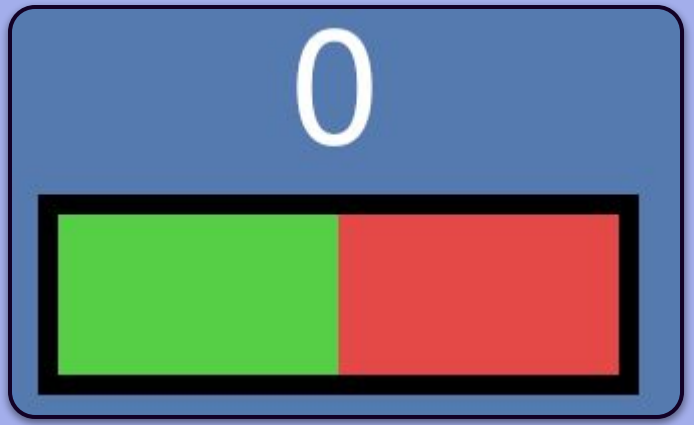
```
1  void OnTriggerEnter(Collider col)
2  {
3      if (col.CompareTag("Borders"))
4      {
5          Vector3 _spawnPosition = _spawnPoint.transform.position;
6          transform.position = new Vector3(_spawnPosition.x, _spawnPosition.y + 0.25f, _spawnPosition.z);
7      }
8  }
```

Détection d'une collision entre notre balle et notre plaine

Modification des fonctions Start et *OnTriggerEnter* de notre script *BallController.cs*

Gestion de l’UI

avec l’ajout du nombre de frappes et de la jauge de force



Implémentation du nombre de frappes

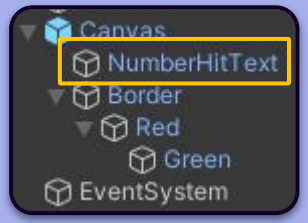
en actualisant l'UI

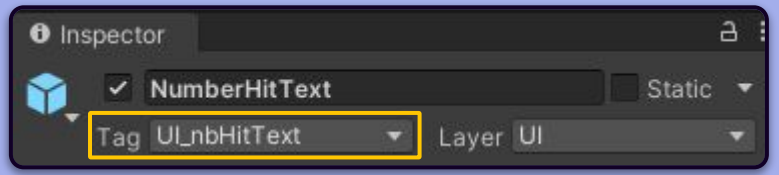
Implémenter le compteur de frappe et sa mise à jour

Vous pouvez passer en mode 2D pour facilement interagir avec votre Canvas

Implémentation du nombre de frappes

en ajoutant un *Text* à notre canvas





Ajout d'un *UI > Legacy > Text*

Ajout du tag UI_nbHitText

Implémentation du nombre de frappes

à partir de la modification du script *BallController.cs* (importer le *package UnityEngine.UI*)

```
private int _nbHit = 0;
private Text _nbHitText;
void Start()
{
    _nbHitText = GameObject.FindGameObjectWithTag("UI_nbHitText").GetComponent<Text>();
}
```

Deux nouveaux attributs

Référence à notre élément *Text*

```
if (Input.GetKeyUp(KeyCode.Space) || _spaceKeyDownDuration > maxTimeSpaceKeyDown)
{
    _spaceUp = true;
    _spaceDown = false;
    Vector3 dir = (this.transform.position - Camera.main.transform.position).normalized;
    _applyForce = dir * baseSpeed * _spaceKeyDownDuration;
    _applyForce.y = 0;
    _nbHit += 1;
    UpdateUi();
}
```

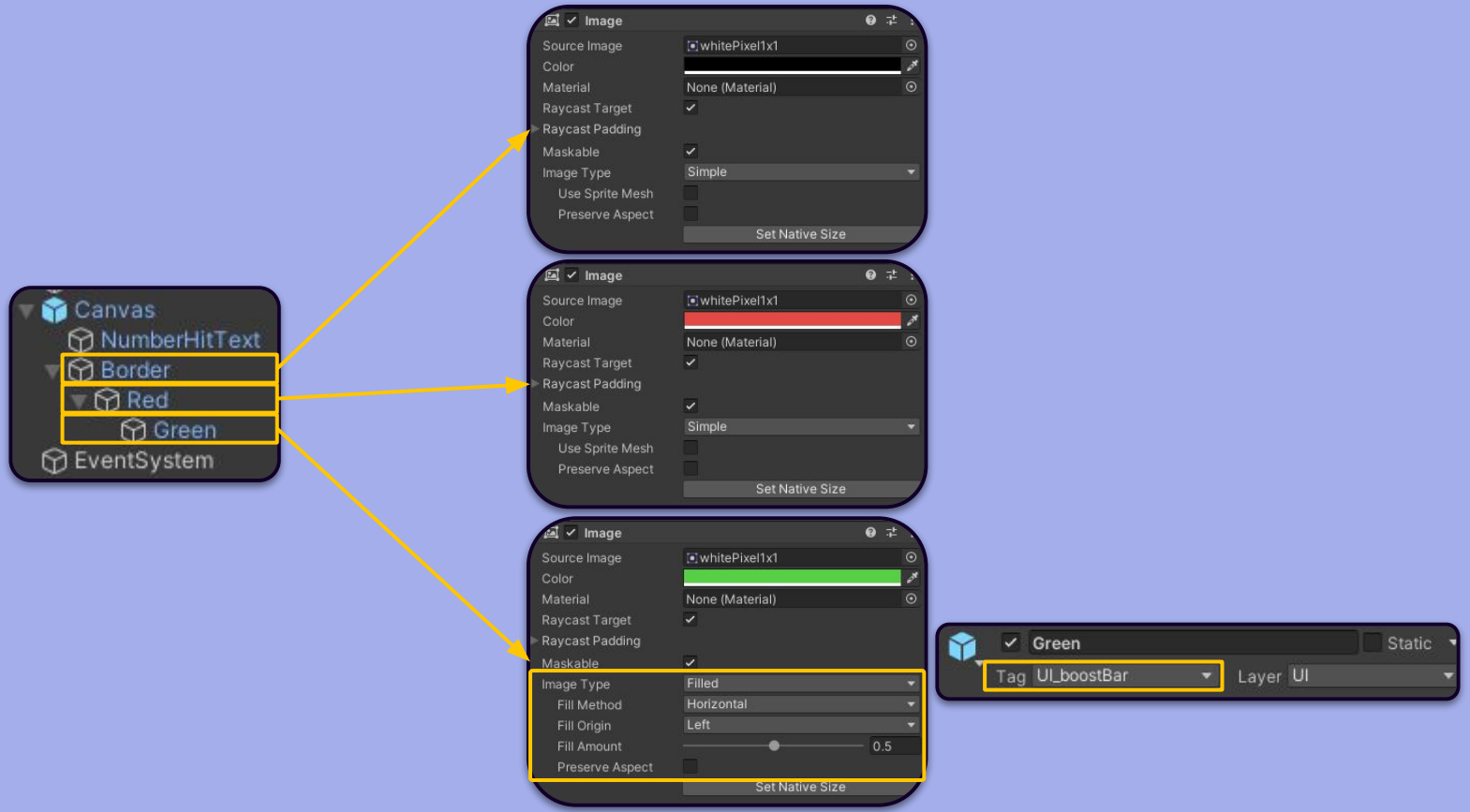
Mise à jour de l'attribut *_nbHit* et appel de la fonction *UpdateUi*

```
void UpdateUi()
{
    _nbHitText.text = _nbHit.ToString();
}
```

Implémentation de la méthode *UpdateUi* permettant de mettre à jour notre *Text UI* avec l'attribut *_nbHit*

Implémentation de la jauge de force

avec la superposition de plusieurs *UI > Image* imbriquées



Implémentation du nombre de frappes

à partir de la modification du script *BallController.cs*

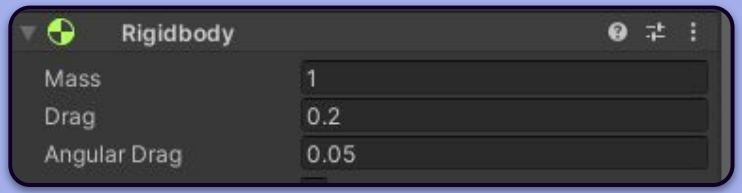
```
private Image _greenBar;
// Start is called before the first frame update
void Start()
{
    _greenBar = GameObject.FindGameObjectWithTag("UI_boostBar").GetComponent<Image>();
    _greenBar.fillAmount = 0;
}
```

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.Space))
    {
        _spaceDown = true;
    }

    if (_spaceDown)
    {
        _spaceKeyDownDuration += Time.deltaTime;

        _greenBar.fillAmount = _spaceKeyDownDuration / maxTimeSpaceKeyDown;
    }

    if (Input.GetKeyUp(KeyCode.Space) || _spaceKeyDownDuration > maxTimeSpaceKeyDown)
    {
        _spaceUp = true;
        _spaceDown = false;
        Vector3 dir = (this.transform.position - Camera.main.transform.position).normalized;
        _applyForce = dir * baseSpeed * _spaceKeyDownDuration;
        _applyForce.y = 0;
        _nbHit += 1;

        _greenBar.fillAmount = 0;

        UpdateUi();
    }
}
```

Peaufiner le projet

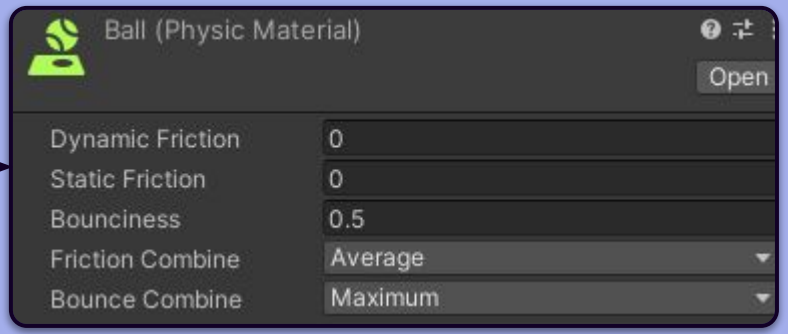
avec les *Physic Material* et la paramétrisation du moteur physique

Ajouter une valeur de friction à votre balle (propriété *Drag* et *Angular Drag*)



Ajouter une valeur de “rebondissement” lorsque la balle rentre en collision

Créer un *Physic Material* et glisser-déposer le au niveau de notre balle

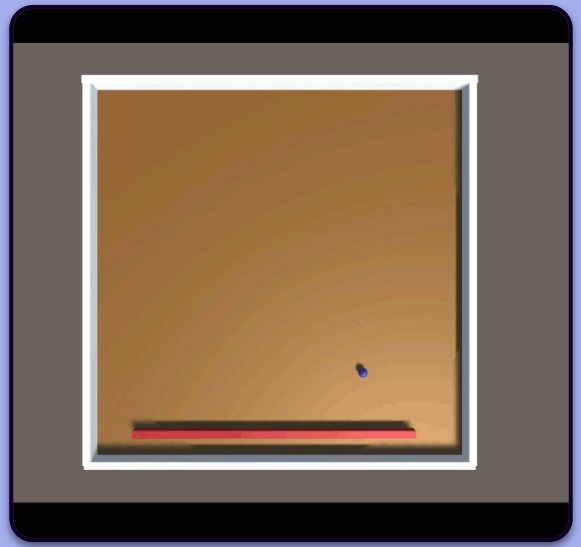


https://forum.unity.com/threads/is-rigibody-drag-same-as-physics-material-dynamic-friction.1144442/

Paramétrer la physique

avec la modification de la propriété *Collision Detection* du *Rigidbody* de la balle

La valeur *Continuous* de la propriété *Collision Detection* du *Rigidbody* de la balle permet d'éviter un effet de "tunneling" lorsque la balle rentre en collision avec un rebord du terrain à une vitesse élevée



https://docs.unity3d.com/Manual/ContinuousCollisionDetection.html

Paramétrer la physique

avec la modification de la valeur *Default Contact Offset* du moteur physique

La modification de la propriété *Edit > Project Settings > Physics > Default Contact Offset* à 0.001 permet d'éviter des "*Ghost collision*" lors de la jonction entre deux tuiles de terrain

https://forum.unity.com/threads/ghost-collisions-on-adjacent-box-colliders.850951/

ProBuilder

Package de modélisation